组成原理实验课程第_二_次实报告

实验名称	乘法器的模拟与上箱实现			班级	李涛
学生姓名	艾明旭	学号	2111033	指导老师	董前琨
实验地点	B306		实验时间	2023 年 4 月 5 日	

1、 实验目的

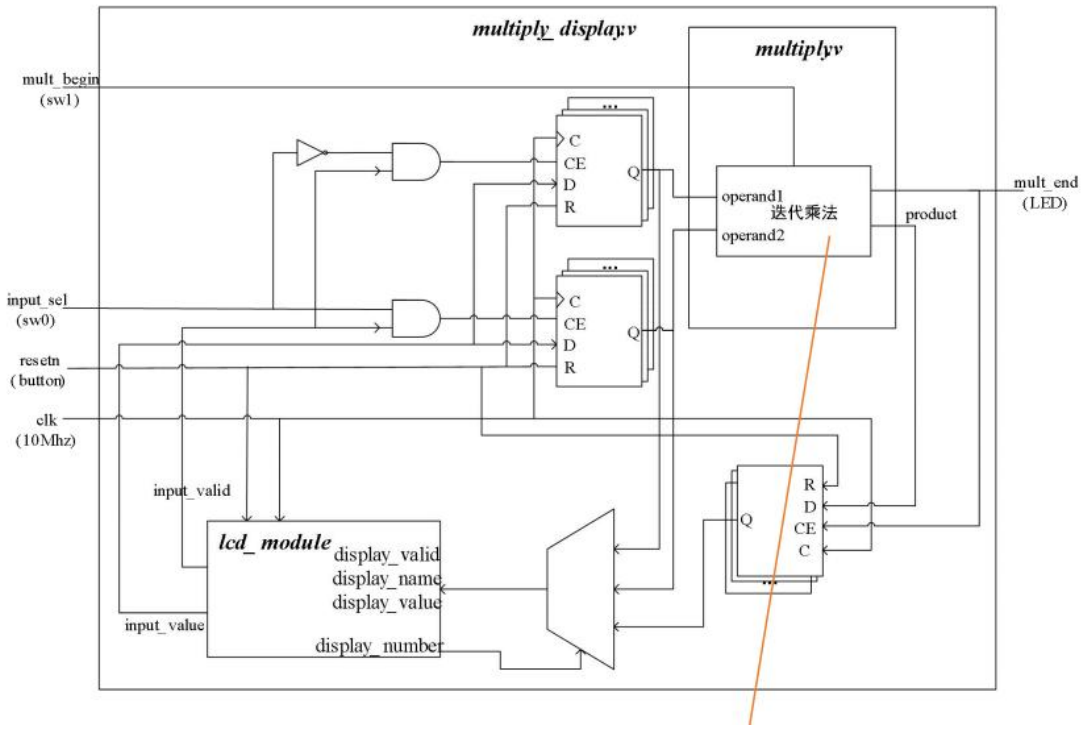
1. 理解定点乘法的不同实现算法的原理，掌握基本实现算法。
2. 熟悉并运用 verilog 语言进行电路设计。
3. 为后续设计 cpu 的实验打下基础。

2、 实验内容说明

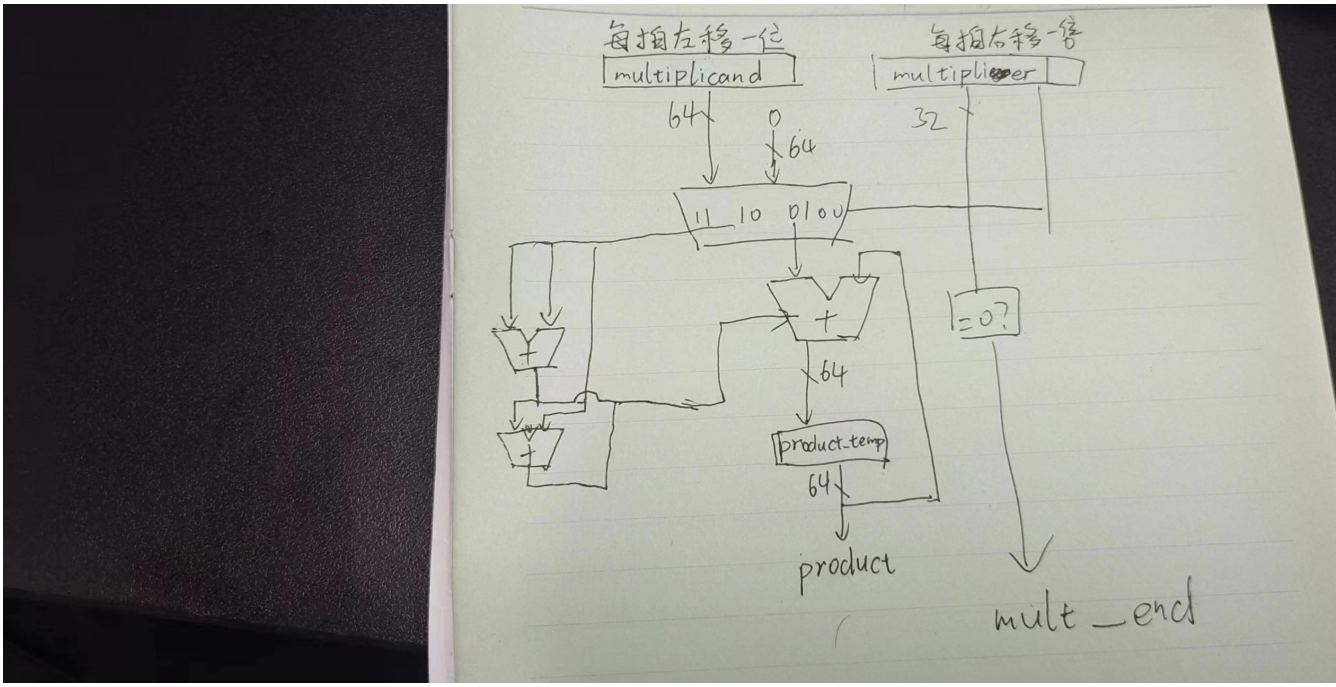
在理解定点乘法的基础上，利用 verilog 语言进行定点乘法的编写，之后对编写模块仿真实验、烧制至主板验证。

在实现 32 位被乘数和乘数在 32 周期内完成乘法运算的基础上，思考如何改进使得在 16、8 周期内完成实验。

3、 实验原理图



4、



4.实验步骤

（分布介绍依次完成了哪些代码修改，从而实现了什么样的功能）

1、修改乘法迭代部分，实现周期数的对数级减少。

```
assign op1_sign = mult_op1[31];
assign op2_sign = mult_op2[31];
assign op1_absolute = op1_sign ? (~mult_op1+1) : mult_op1;
assign op2_absolute = op2_sign ? (~mult_op2+1) : mult_op2;

//加载被乘数，运算时每次左移二位
reg  [63:0] multiplicand;
always @ (posedge clk)
begin
    if (mult_valid)
    begin    // 如果正在进行乘法，则被乘数每时钟左移二位
        multiplicand <= {multiplicand[61:0],2'b00};
    end
    else if (mult_begin)
    begin   // 乘法开始，加载被乘数，为乘数1的绝对值
        multiplicand <= {32'd0,op1_absolute};
```

```
//加载乘数，运算时每次右移二位
reg  [31:0] multiplier;
always @ (posedge clk)
begin
    if (mult_valid)
    begin   // 如果正在进行乘法，则乘数每时钟右移二位
        multiplier <= {2'b00,multiplier[31:2]};
    end
    else if (mult_begin)
    begin   // 乘法开始，加载乘数，为乘数2的绝对值
        multiplier <= op2_absolute;
    end
end
```

```
// 部分积：乘数末位为1，由被乘数左移得到；乘数末位为0，部分积为0
wire [63:0] partial_product;
assign partial_product = multiplier[1:0]==2'b11 ? multiplicand + multiplicand + multiplicand:
                         multiplier[1:0]==2'b10 ? multiplicand + multiplicand:
                         multiplier[1:0]==2'b01 ? multiplicand : 64'd0;

//累加器
reg [63:0] product_temp;
always @ (posedge clk)
begin
    if (mult_valid)
    begin
        product_temp <= product_temp + partial_product;
    end
    else if (mult_begin)
```

如上图，图 1 在每一次时钟的上升沿使得被乘数左移两位，乘数右移两位，图 2 则因为每次看乘数最后两位二进制，若 11，则 partial_product*3,若 10，则 partial_product*2,01，则 partial_product*1，00，则 0，最后叠加到结果即可，叠加部分无需改变。

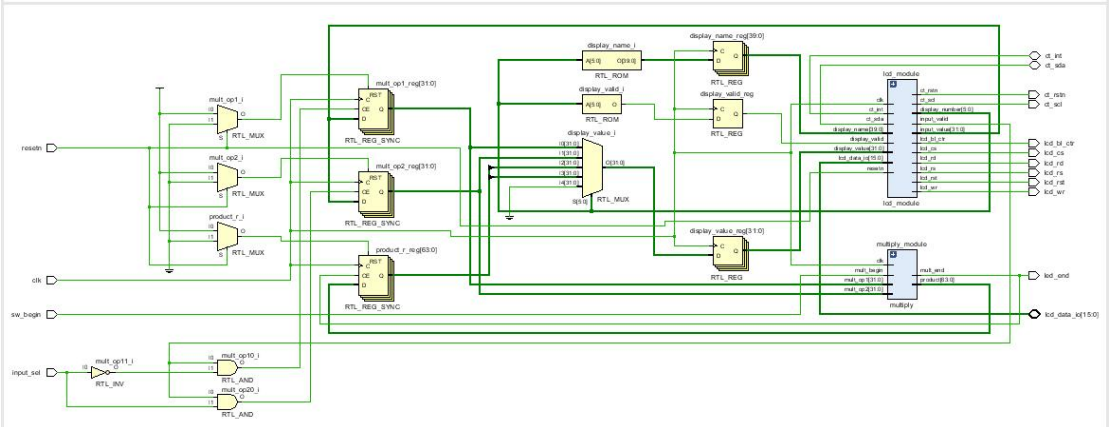
2、修改触摸屏的显示位置

```
case(display_number)
    6'd5 :
    begin
        display_valid <= 1'b1;
        display_name  <= "M_OP1";
        display_value <= mult_op1;
    end
    6'd9 :
    begin
        display_valid <= 1'b1;
        display_name  <= "M_OP2";
        display_value <= mult_op2;
    end
    6'd13 :
    begin
        display_valid <= 1'b1;
        display_name  <= "PRO_H";
        display_value <= product_r[63:32];
    end
    6'd17 :
    begin
```

此处将 case 语句下面的分类中的数字对应修改即可。

模拟图如下：

实验门电路及引脚图如下：



5、 实验结果分析

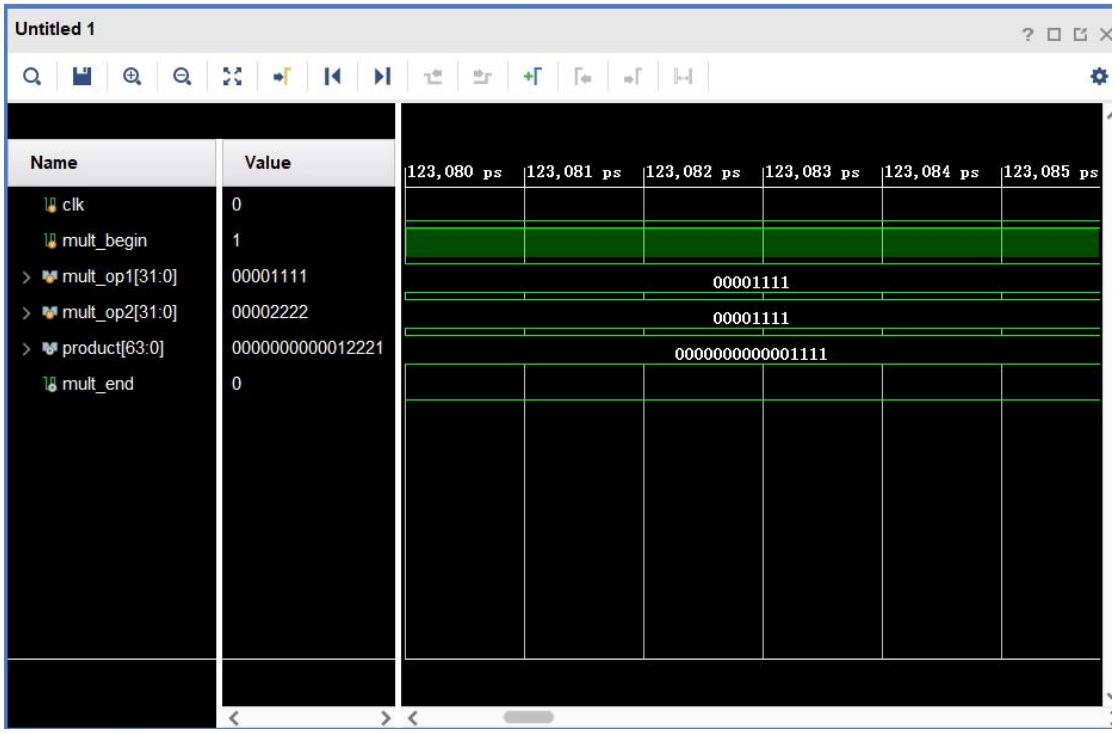
（仿真结果截图或者实验箱运行结果拍照，注意需要对实验结果进行分析，输入是什么，输出是什么，结果是什么，是否验证了正确性）

仿真截图：

时钟周期 1 product 为 0

时钟周期 2 第一个乘法周期，对于 0001 0001 0001 0001 * 0001 0001 0001 0001

按照上述乘法规则，乘数末尾 01，进行一次相加，得到 0x1111 正确



时钟周期 3

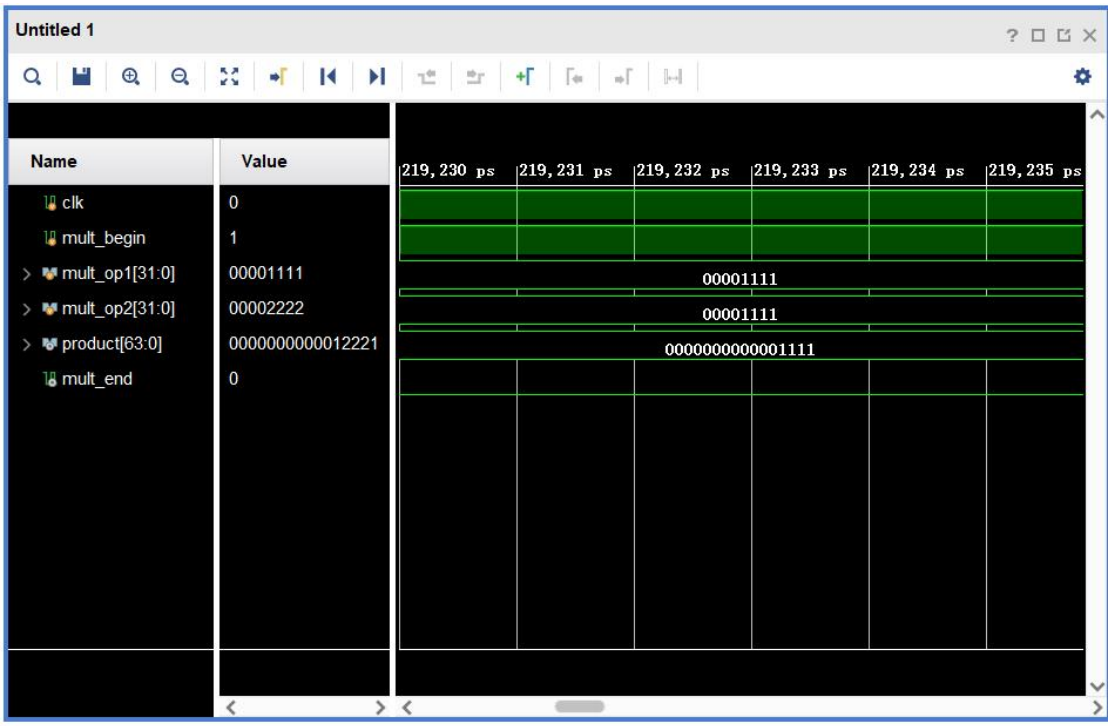
1111 左移 2，1111 右移 2，

即为 0001 0001 0001 0001 00 * 0001 0001 0001 00

乘数末尾 00 ， partial_product 不改变

再加上一周期的 0001 0001 0001 0001

Product 还不变



时钟周期 4 1111 左移 4，1111 右移 4，

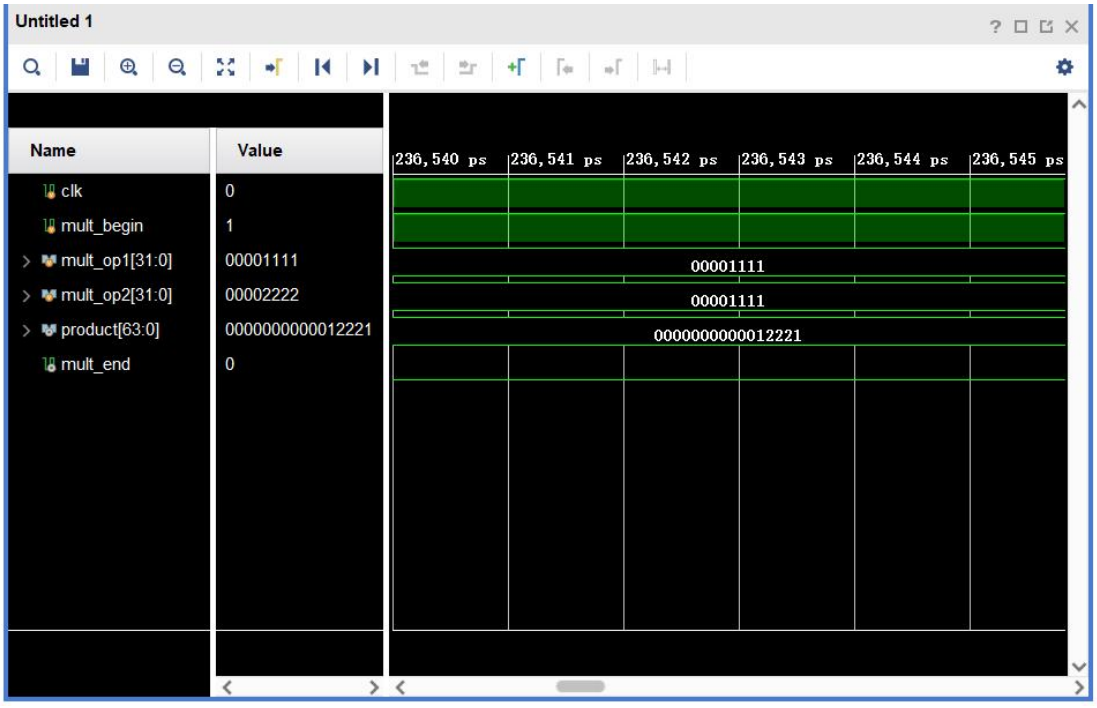
即为 0001 0001 0001 0001 0000 * 0001 0001 0001

乘数末尾 01 ， partial_product 增加一倍的被乘数

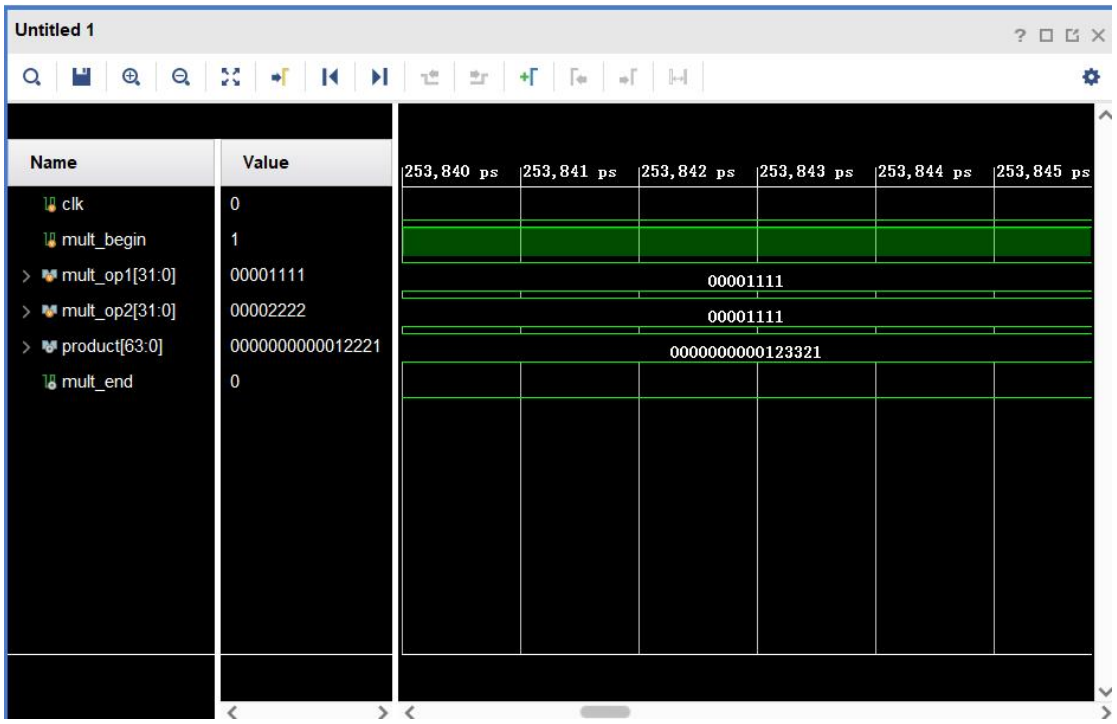
再加上一周期的 0001 0001 0001 0001

即 0x11110 + 0x1111 = 0x12221

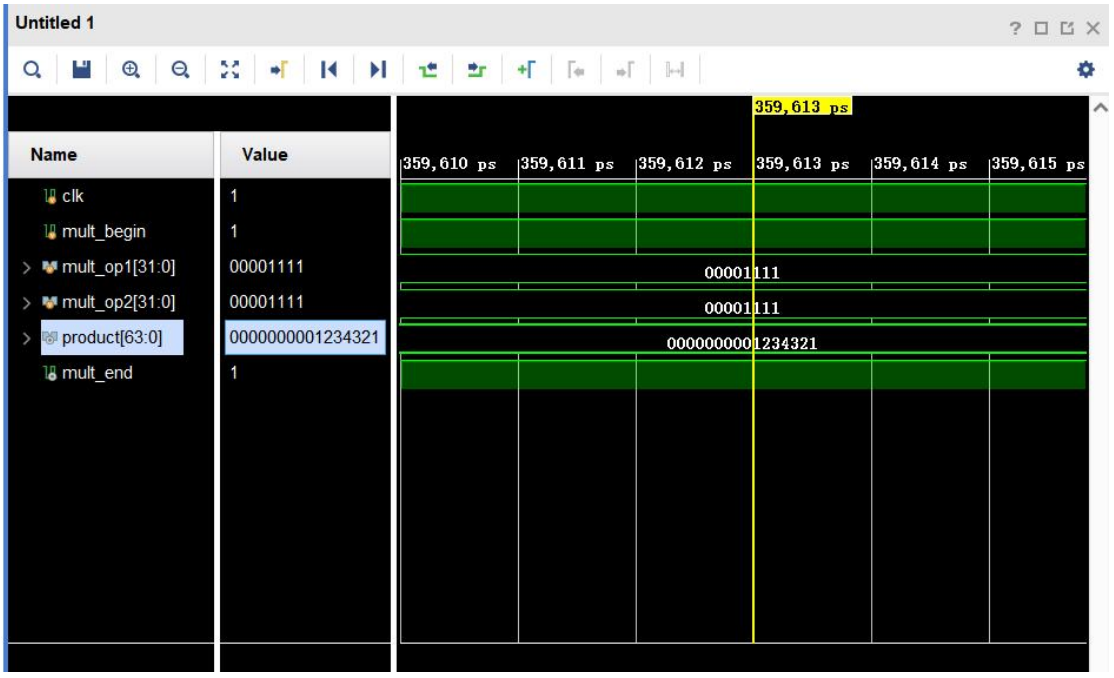
预得到的 pro 与下面图片的结果一致



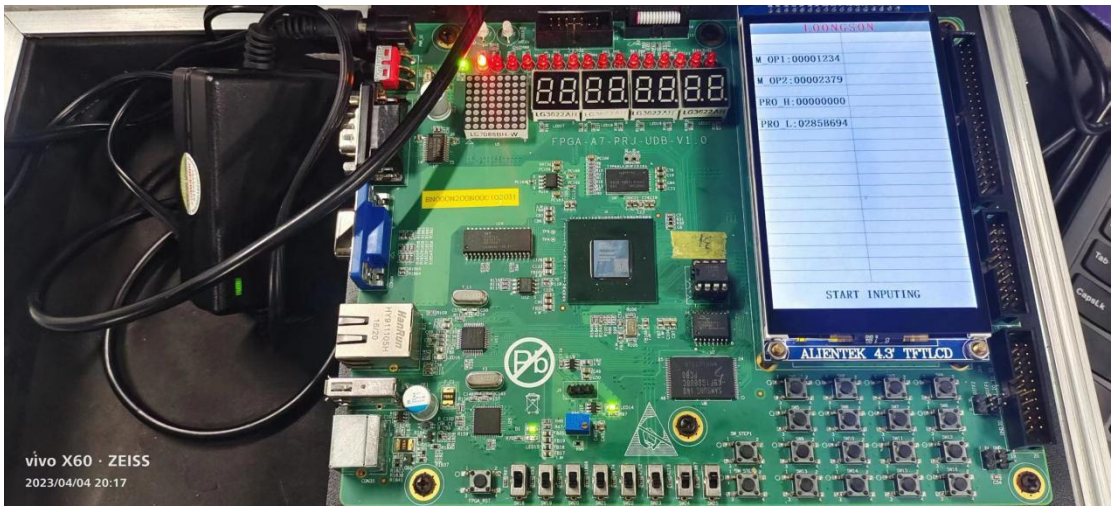
后续时钟 5-8 类似上述对比都正确，看周期 7 的结果也为得到的正确值。



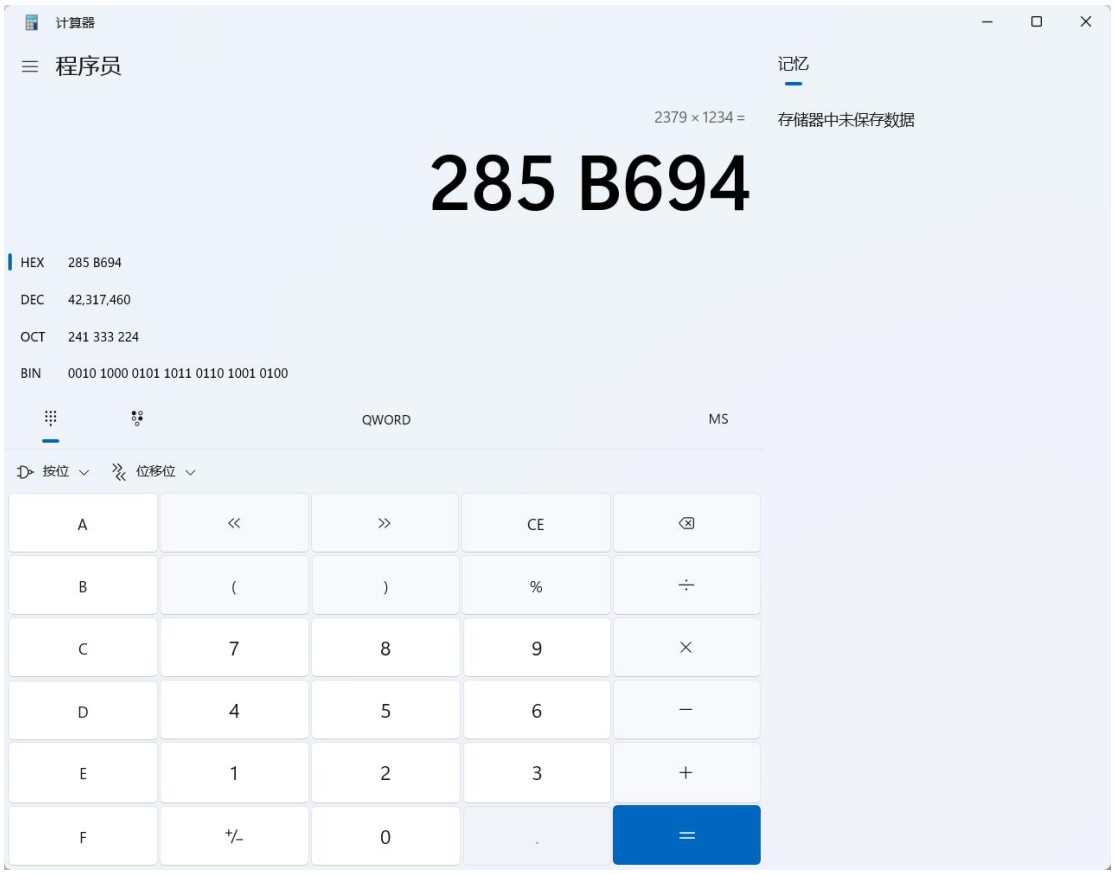
最后一个周期得到正确的结果

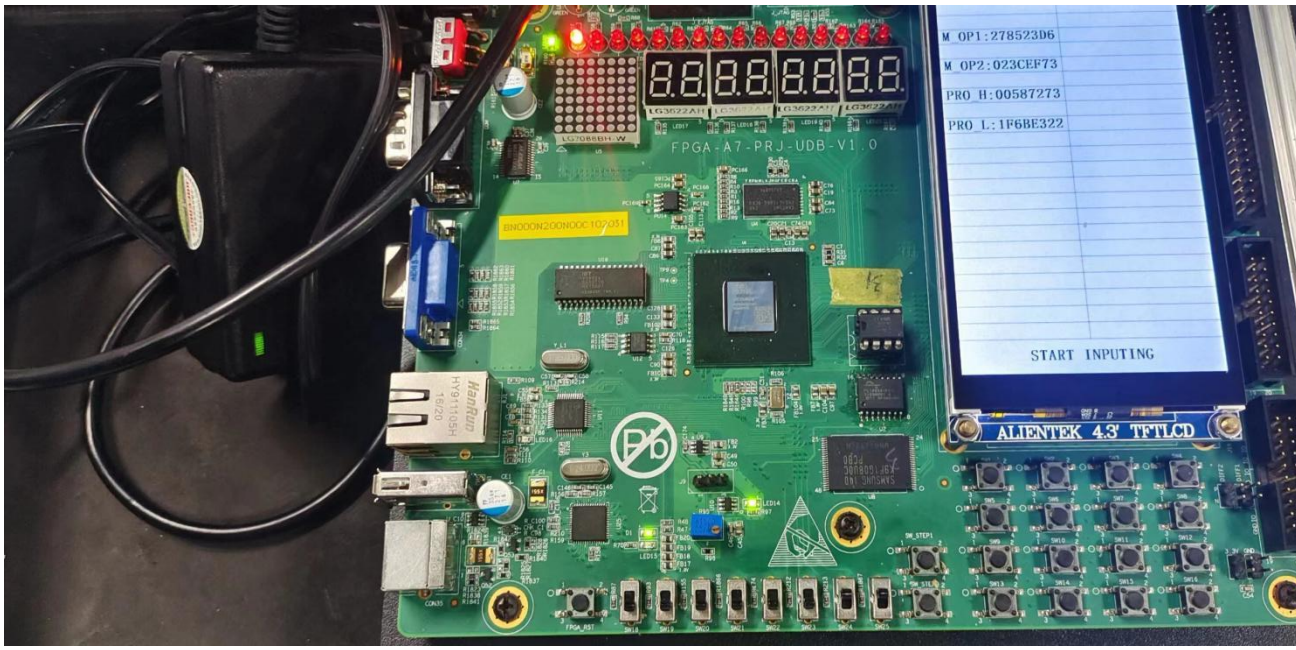


实验箱截图

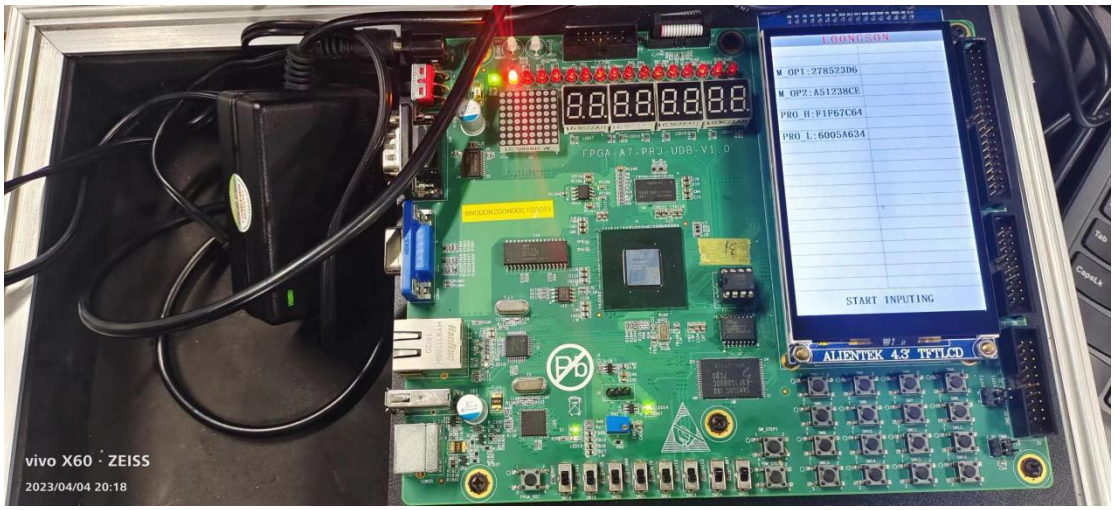


用电脑计算器进行验证，可以得到正确的结果。





当数值较大而不影响首位数字时，实验结果的正确性仍然不受影响。



而当我们真正影响到了第一位的存储数值，由于第一位是符号位，说明存储的是负值，需要转化为补码之后进行计算，而不能直接计算，如果得到的结果也为负值，同样也会按补码的方式存储。

6.总结感想

这次实验，让我体会到计算机内部如何巧妙实现、并且优化乘法指令，利用其被乘数和乘数的移位，巧妙迭代 partial_product 将乘法完成。也使得我对 verilog 语言更加熟悉，能够熟练上机仿真。

同时，我还对实验仿真和程序当中的数的存储有了一定的认识，因为存储的是补码，所以当我们输入的数的最高位是 1 的时候，我们会得到一个负数，然后计算的结果也是按这个数为负数来计算，最终的到的也会是按此方式存储的数。

同时，我还学会了利用两位移位进行的数据乘算和移位操作，还对数据的末尾对数据的乘法运算的影响有了一定的了解。

最后，我还学会了乘法器的迭代运行方式，能够较为熟练的描述乘法器的运行模式。